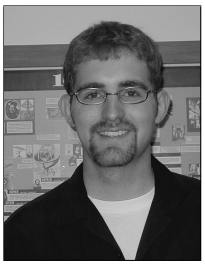
Breaking The Code

Ryan Lowe



Ryan Lowe is currently a Ball State senior with a double major in Computer Science and Mathematics and a minor in Applied Physics. As a sophomore, he took an independent study course on cryptography, where he learned the abstract algebra behind various encryption techniques, such as RSA public key cryptography and elliptic curve cryptography.

Introduction

The RSA public key code [1, Chapter 12] is the most widely used encryption technique in modern communication networks, such as the Internet. Its security relies on the fact that no known (classical) computer program can *efficiently* factor large numbers n, which are the product of two (unknown) primes, p_1 and p_2. Quantum computers, which currently exist only as mathematical models and have not yet been physically built, can perform certain tasks with efficiency that is not matched by standard computers. In this article we will learn how, in theory, one such task can be used to factor n in a reasonable amount of time, thus breaking the RSA code. I studied this algorithm as part of an independent study cryptography course, which I took with Dr. Fischer. This article is a summary of a talk I gave on the subject for the undergraduate colloquium series in August 2002.

The Algorithm

Given the product n of two distinct odd prime numbers p_1 and p_2, the goal of the algorithm is to efficiently find p_1 and p_2, and it attempts to do so via a certain randomized process. It will suffice, if the algorithm produces the desired output in 50% of all cases, since repeated application will then quickly result in success.

This algorithm, which runs on the current mathematical models of quantum computers, was first conceived by Shor [4]. Our exposition is modeled on Preskill [3]. Here is the pseudo code:

Input: A product n of two distinct (unknown) odd prime numbers p_1 and p_2.

Execute the following routine as often as needed until n has been factored:

Step 1. Randomly, select a number $a \in \{1, 2, \cdots, n-1\}$.

Step 2. Compute the greatest common divisor $\gcd(a, n)$ of a and n.

Step 3. If $\gcd(a, n) \neq 1$, then output "$p_1 = a$ and $p_2 = n/a$" and stop; else continue.

Step 4. [This step requires a quantum computer.] Find the smallest positive integer r with $a^r \equiv 1 \pmod{n}$.

Step 5. If r is odd, then output "factoring failed in this round" and stop.

Step 6. Compute $q = \gcd(a^{r/2} + 1, n)$. If $q = n$, then output "factoring failed in this round" and stop; else output "$p_1 = q$ and $p_2 = n/q$" and stop.

Analysis of the Algorithm

First, we randomly select a number a between 1 and $n-1$. Next, we use the very efficient Euclidean Algorithm [1, Theorem 1.6] to compute the greatest common divisor $\gcd(a, n)$ of a and n. If $\gcd(a, n) \neq 1$, then $a = p_1$ or $a = p_2$, because the only divisors of n are $1, p_1, p_2$ and n. We will assume the worst case, where $\gcd(a, n) = 1$, and proceed to Step 4.

Note that the set $\mathbb{Z}_n^* = \{k \pmod{n} \mid \gcd(k, n) = 1\}$ forms a group under multiplication [1, Corollary 2.10]. Since this group is finite, there is a smallest positive integer r, called the order of $a \pmod{n}$, such that $a^r \equiv 1 \pmod{n}$[1, Theorem 7.8(1)].

The next step of the algorithm calls for computing the value of r, which can be done by computing the period of the function $f(x) = a^x$, since

$$f(x + r) = a^{x+r} = a^x \cdot a^r = a^x \cdot 1 = f(x) \pmod{n}.$$

It is not feasible to carry out this step on a classical computer. However, a quantum computer is able to find this period (in theory) in an efficient manner. (See [2] to learn how.)

If r turns out to be odd, our algorithm fails. From here on, we are going to assume that r is even. Let us denote the phrase "i divides j" by $i|j$. Since $a^r \equiv 1 \pmod{n}$, then

$$n \mid (a^r - 1) = (a^{r/2} - 1) \cdot (a^{r/2} + 1).$$

We know that $n \nmid (a^{r/2} - 1)$, otherwise r would not be the order of a. (If $n \mid (a^{r/2} - 1)$, then $a^{r/2} \equiv 1 \pmod{n}$, contradicting the choice of r.) Therefore, $\gcd(a^{r/2} + 1, n) \neq 1$. Now if $n \nmid (a^{r/2} + 1)$, i.e. if $a^{r/2} \not\equiv -1 \pmod{n}$, then n must have a nontrivial common factor with each of $a^{r/2} \pm 1$, which we extract by computing[1] $\gcd(a^{r/2} + 1, n) \in \{p_1, p_2\}$.

So we succeed in factoring n, unless either r is odd, or r is even and $a^{r/2} \equiv -1 \pmod{n}$.

[1] A computational note: $\gcd(a^{r/2} + 1, n)$ is found by first reducing $a^{r/2}$ modulo n. Even for very large numbers, $a^{r/2} \pmod{n}$ can be computed efficiently. This is done by first expressing the exponent $r/2$ as the sum of powers of 2 and then repeatedly squaring a, each

How Likely Is Success?

Recall that $n = p_1 \cdot p_2$, where p_1 and p_2 are two distinct odd primes. By the Chinese Remainder Theorem (CRT) [1, Theorem 13.2], for each pair a_1 and a_2 of integers with $0 \leq a_1 < p_1$ and $0 \leq a_2 < p_2$, there exists exactly one integer a with $0 \leq a < p_1 \cdot p_2$, such that

$$a \equiv a_1 \pmod{p_1}$$

$$a \equiv a_2 \pmod{p_2}$$

Hence, randomly choosing $a \in \{1, 2, \cdots, n-1\}$ with $\gcd(a, n) = 1$, is equivalent to randomly choosing $a_1 \in \{1, 2, \cdots, p_1 - 1\}$ and $a_2 \in \{1, 2, \cdots, p_2 - 1\}$, independently. (Note that $a_i = 0$ corresponds to $p_i | a$.)

Now, let r_1 denote the order of $a_1 \pmod{p_1}$ and let r_2 denote the order of $a_2 \pmod{p_2}$. By the CRT, $a^x \equiv 1 \pmod{p_1 \cdot p_2}$ is equivalent to

$$a^x \equiv a_1^x \equiv 1 \pmod{p_1}$$

$$a^x \equiv a_2^x \equiv 1 \pmod{p_2}$$

This, in turn, is equivalent to $r_1 \mid x$ and $r_2 \mid x$ [1, Theorem 7.8(3)]. Consequently, since r is the smallest positive x with $a^x \equiv 1 \pmod{p_1 \cdot p_2}$, then $r = \mathrm{lcm}(r_1, r_2)$. Therefore, if both r_1 and r_2 are odd, then so is r and the factoring attempt fails. If either r_1 or r_2 is even, then so is r and we proceed.

Since $a^r \equiv a_i^r \equiv 1 \pmod{p_i}$, we see that $p_i \mid a^r - 1 = (a^{r/2} + 1) \cdot (a^{r/2} - 1)$. However, p_i is prime, so that either $p_i \mid (a^{r/2} + 1)$ or $p_i \mid (a^{r/2} - 1)$. In other words:

$$a^{r/2} \equiv \pm 1 \pmod{p_1}$$

$$a^{r/2} \equiv \pm 1 \pmod{p_2}$$

We will now analyze the four cases:

Case 1	Case 2	Case 3	Case 4
$a^{r/2} \equiv 1 \bmod p_1$	$a^{r/2} \equiv -1 \bmod p_1$	$a^{r/2} \equiv -1 \bmod p_1$	$a^{r/2} \equiv \ \ 1 \bmod p_1$
$a^{r/2} \equiv 1 \bmod p_2$	$a^{r/2} \equiv -1 \bmod p_2$	$a^{r/2} \equiv \ \ 1 \bmod p_2$	$a^{r/2} \equiv -1 \bmod p_2$
$a^{r/2} \equiv 1 \bmod n$	$a^{r/2} \equiv -1 \bmod n$	$a^{r/2} \not\equiv -1 \bmod n$	$a^{r/2} \not\equiv -1 \bmod n$
Not Possible	Failure	Success	Success

Let us factor out all of the 2's from r_1 and r_2, that is, let us write

$$r_1 = 2^{C_1} \cdot m_1$$

$$r_2 = 2^{C_2} \cdot m_2$$

with two odd integers m_1 and m_2 and $C_i \geq 0$.

time reducing the result modulo n. Multiplying the appropriate powers $a^{(2^i)} \pmod{n}$, while reducing modulo n after each multiplication, finally yields $a^{r/2} \pmod{n}$. This procedure decreases the number of necessary arithmetic steps on a logarithmic scale.

Suppose now that $C_1 > C_2$, then $r = \text{lcm}(r_1, r_2) = 2 \cdot r_2 \cdot z$, for some integer z, and we claim that $a^{r/2} \equiv 1 \pmod{p_2}$ and $a^{r/2} \equiv -1 \pmod{p_1}$, which leads us to Case 3. To see this, first note that $a^{r/2} \equiv (a^{r_2})^z \equiv (a_2^{r_2})^z \equiv 1^z \equiv 1 \pmod{p_2}$. And since $a^{r/2} \not\equiv 1 \pmod{n}$, we must have $a^{r/2} \not\equiv 1 \pmod{p_1}$. So we obtain $a^{r/2} \equiv -1 \pmod{p_1}$.

By symmetry, if $C_2 > C_1$, we end up in Case 4.

Now let us suppose that $C_1 = C_2 \geq 1$. Then $r = \text{lcm}(r_1, r_2) = r_1 \cdot u_1 = r_2 \cdot u_2$, for some *odd* integers u_i. We claim that $a^{r/2} \equiv -1 \pmod{p_1}$ and $a^{r/2} \equiv -1 \pmod{p_2}$, which places us into Case 2. To see this, first notice that $r_1 \nmid \frac{r_1}{2} \cdot u_1$, due to the lack of a factor of 2 on the right hand side. Hence, $a^{r/2} = a_1^{\frac{r_1}{2} \cdot u_1} \not\equiv 1 \pmod{p_1}$. By symmetry we also get $a^{r/2} = a_2^{\frac{r_2}{2} \cdot u_2} \not\equiv 1 \pmod{p_2}$.

Moreover, r will be odd if and only if $C_1 = C_2 = 0$, in which case the algorithm fails.

In summary, we succeed if and only if $C_1 \neq C_2$.

We now wish to estimate the likelihood of this to occur. To this end, we view C_i as a function of the variable a_i, whose domain is the set $\{1, 2, \cdots, p_i - 1\}$. Let $\hat{C}_i$ denote the maximum value of the function C_i. Below, we will show that the function C_i assumes its maximum value $\hat{C}_i$ for exactly 50% of all $a_i \in \{1, 2, \cdots, p_i - 1\}$. For now, let us assume this to be true. Then each of the following four systems of equations is true exactly 25% of the time:

(1) $C_1 = \hat{C}_1$ and $C_2 = \hat{C}_2$;
(2) $C_1 = \hat{C}_1$ and $C_2 < \hat{C}_2$;
(3) $C_1 < \hat{C}_1$ and $C_2 = \hat{C}_2$;
(4) $C_1 < \hat{C}_1$ and $C_2 < \hat{C}_2$.

If $\hat{C}_1 = \hat{C}_2$, we succeed in Situations (2) and (3). If $\hat{C}_1 < \hat{C}_2$, we succeed in Situations (1) and (3). Finally, if $\hat{C}_1 > \hat{C}_2$, we succeed in Situations (1) and (2). So, either way, we succeed in at least 50% of all cases—as we claimed we would.

The remaining task is to verify the following

Lemma. *$C_i = \hat{C}_i$ for exactly half of the values of $a_i \in \{1, 2, \cdots, p_i - 1\}$.*

PROOF. Recall that the multiplicative group

$$\mathbb{Z}_{p_i}^* = \{1 \pmod{p_i}, 2 \pmod{p_i}, \cdots, p_i - 1 \pmod{p_i}\}$$

is cyclic [1, Theorem 7.15] and therefore must contain a generating element of order $p_i - 1$. Let $b_i \pmod{p_i}$ be such a generator, that is, suppose

$$\mathbb{Z}_{p_i}^* = \{b_i^1 \pmod{p_i}, b_i^2 \pmod{p_i}, \cdots, b_i^{p_i - 1} \pmod{p_i}\}.$$

Say, $a_i \pmod{p_i} = b_i^{t_i} \pmod{p_i}$. Recall that r_i is defined to be the order of the element $a_i \pmod{p_i}$. Now, $a_i^x \equiv b_i^{t_i \cdot x} \equiv 1 \pmod{p_i}$ if and only $p_i - 1 | (t_i \cdot x)$, which is in turn equivalent to

$$\frac{p_i - 1}{\gcd(t_i, p_i - 1)} \mid x.$$

Therefore,

$$2^{C_i} \cdot m_i = r_i = \text{order}(a_i) = \frac{p_i - 1}{\gcd(t_i, p_i - 1)}.$$

Writing $p_i - 1 = 2^{k_i} \cdot s_i$ with $k_i \geq 1$ and some odd integer s_i, we see that $\hat{C}_i = k_i$ and that $C_i = k_i$ if and only if t_i is odd. This happens half of the time. □

Conclusion

We have presented an algorithm, which successfully factors $n = p_1 \cdot p_2$ at least 50% of the time and does so efficiently, if implemented on a quantum computer. Note that repeated execution of this algorithm will result in rapidly increasing probability of success. Here is a table of accumulative failure probabilities:

Number of Executions	Probability of Not Having Factored n
1	$(1/2)^1 = .500$
2	$(1/2)^2 = .250$
3	$(1/2)^3 = .125$
$\vdots$	$\vdots$
7	$(1/2)^7 \approx .008$

Already after 7 executions, this algorithm has a 99.2% chance of succeeding in factoring n.

References

[1] T.W. Hungerford, Abstract Algebra: An Introduction (Second Ed.), Saunders College Publishing (1997).

[2] A.O. Pittenger, An introduction to quantum computing algorithms, Birkhäuser (2001).

[3] J. Preskill, *Quantum Information and Computation*, Lecture notes for Physics 229, Available at `http://www.theory.caltech.edu/~preskill`

[4] P. Shor, *Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer*, SIAM J. Comput. **26** (1997) 1484–1509.